Learning-Based Model Predictive Control for Autonomous Racing

João Pinho 1,*, Gabriel Costa 1, Pedro U. Lima 1 and Miguel Ayala Botto 2

1 Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal; gaf@live.com.pt (G.C.); pedro.lima@tecnico.ulisboa.pt (P.U.L.)
2 IDMEC, Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal; ayalabotto@tecnico.ulisboa.pt

* Correspondence: joao.g.pinho@tecnico.ulisboa.pt

Abstract: In this paper, we present the adaptation of the terminal component learning-based model predictive control (TC-LMPC) architecture for autonomous racing to the Formula Student Driverless (FSD) context. We test the TC-LMPC architecture, a reference-free controller that is able to learn from previous iterations by building an appropriate terminal safe set and terminal cost from collected trajectories and input sequences, in a vehicle simulator dedicated to the FSD competition. One major problem in autonomous racing is the difficulty in obtaining accurate highly nonlinear vehicle models that cover the entire performance envelope. This is more severe as the controller pushes for incrementally more aggressive behavior. To address this problem, we use offline and online measurements and machine learning (ML) techniques for the online adaptation of the vehicle model. We test two sparse Gaussian process regression (GPR) approximations for model learning. The novelty in the model learning segment is the use of a selection method for the initial training dataset that maximizes the information gain criterion. The TC-LMPC with model learning achieves a 5.9 s reduction (3%) in the total 10-lap FSD race time.

Keywords: electric vehicle; model predictive control; learning-based control; model learning for control; Gaussian processes; motion and path planning; autonomous racing; Formula Student

1. Introduction

Autonomous driving (AD) is an active field of research for academia and the industry. Autonomous driving cars are of growing importance nowadays, and this is expected to increase in the future. Most stakeholders aim at improving road safety [1], reducing traffic congestion and emissions [2], and deploying universal mobility. The applications span from personal vehicles, robotaxis, and delivery vehicles to long-haul trucks, shuttles, and mining vehicles [3–6].

This paper focuses on autonomous electric racing, a subfield of autonomous driving that aims to contribute to the broader problem by introducing innovations in autonomous technology through sport [7]. This type of synergy is well established between Formula One and Formula E and the automotive industry. In particular, we aim to contribute to the tasks of trajectory planning and control.

Paden et al. [8] surveyed planning and control algorithms for AD in the urban setting. Several controllers that resort to a kinematic bicycle model have been designed, e.g., the Stanley controller [9]. To handle more demanding driving manoeuvres, more complex controllers must be designed. Advances in computing hardware and mathematical programming have made model predictive control (MPC) feasible for real-time use in AD [10].

We target the 10-lap trackdrive event of the Formula Student Driverless (FSD) autonomous racing competition. Formula Student (www.formulastudent.de/about/concept/ (accessed on 20 April 2023)) is a student engineering competition where teams design and compete with a formula race car. The autonomous racing competition environment is
controlled in the sense that no other agents, such as other vehicles or pedestrians, are near the track. The track is composed of blue and yellow cones on the left and right borders, respectively. The perception and decision modules in this setting are simpler than in urban AD. In contrast, the motion planning and control problem can be greater as teams aim to drive at the limits of handling [11,12].

Obtaining a sufficiently accurate vehicle model in these conditions without rendering the MPC computationally intractable is a challenging task. Moreover, the recursive feasibility of optimization-based problems is still a challenging issue with respect to their real-time implementation due to the limited computing capacity of processors [13,14]. To overcome this issue, machine learning (ML) techniques have been used to improve the formulation of the MPC using collected data. A recent survey of learning-based model predictive control (LMPC) applications [15] divides the field into the three following categories: (i) system dynamics learning, using ML techniques as a data-based adaptation of the prediction model or uncertainty description (also referred to as model learning); (ii) controller design learning, which targets an MPC controller’s parameterization, e.g., the cost function or terminal components; (iii) MPC for safe learning, which concerns approaches where MPC is used as a safety filter for learning-based controllers.

A cautious LMPC that combines a nominal model with Gaussian process regression (GPR) techniques to model the unknown dynamics was presented in [16]. It has been applied to trajectory tracking with a robotic arm [17]. Alternatively, Bayesian linear regression (BLR) has been used to model the unknown dynamics [18]. The authors argue that this simple model is more accurate in estimating the mean behavior and model uncertainty than GPR and generalizes to novel operating conditions with little or no tuning. Further, they propose a framework that combines BLR model learning with cost learning [19]. However, the algorithm was tested on a slow-moving robot in off-road terrain. Therefore, the main challenge is related to changing dynamics throughout different parts of the terrain.

An LMPC for autonomous racing was applied to the AMZ Driverless FSD prototype [20]. The proposed formulation considers a nominal vehicle model where GPR models residual model uncertainty. The approach is based on model predictive contouring control (MPCC) [21] and cautious MPC [16]. However, the focus is placed only on learning a more accurate vehicle model. We augment this approach by also using the safe set.

The main contribution of this work is the adaptation of Rosolia et al.’s [22] terminal component LMPC (TC-LMPC) architecture to the full-sized autonomous electric racing FSD competition, shown in Figure 1. The same authors present an adaptation to the AD problem [23]; however, it was tested in a miniaturized 1/10 race car. The increased velocities of the full-sized application require a higher control frequency; for this reason, we implement a computationally efficient solution resorting to C++ and FORCESPRO [24,25]. Further, Rosolia et al.’s [23] approach considers tracks with segment-wise constant curvature, while the formulation here presented considers more complex tracks, such as those that we encounter in FSD competitions. Finally, with respect to vehicle modeling, in [23], a linear model is learned using a local linear regressor, while we use the nonlinear version of the bicycle model. Without these adaptations, the completion of a single lap in the full-sized racing application would not be possible. Moreover, we extend the TC-LMPC approach with GPR for model learning [26] by actively choosing the initial training set [27]. We call the combination of the TC-LMPC architecture with ML for model learning TC-LMPCML. We show that TC-LMPC reduces the best lap time by 10% for the 10-lap FSD race. In the work of Kabzan et al. [20], a 10% lap time reduction was also achieved. However, when we activate the model learning component, we show that a 10% lap time reduction can be sustained throughout the event but with a total race time reduction of 3% when compared to the TC-LMPC without the model learning part.

This paper is organized as follows. In Section 2, we provide the theoretical background of the methods used in the LMPC architecture, including the original TC-LMPC formulation. In Section 3, we describe the TC-LMPC architecture for autonomous racing, which improves performance by learning the terminal components (safe set and terminal cost), as well as our
adaptation to the full-sized autonomous electric racing FSD competition—controller design learning. Section 4 explains how the TC-LMPC architecture uses GPR for system dynamics learning. Finally, we introduce the implementation details and show the simulation results for model learning and for both architectures in Section 5. Section 6 provides a summary of the main contributions and suggestions for future research directions.

Figure 1. FST10d—autonomous electric race car from FST Lisboa (©FSG—Cornelius Mosch).

2. Learning-Based Model Predictive Control

We use bold lowercase letters for vectors $x \in \mathbb{R}^n$ and bold capitalized letters for matrices $X \in \mathbb{R}^{n \times m}$, while scalars are non-bold.

2.1. Model Predictive Control

The idea of receding horizon control (RHC) is that an infinite-horizon sub-optimal controller can be designed by repeatedly solving finite-time constrained optimal control (FTCOC) problems in a receding horizon fashion [28]. At each sampling time, starting at the current state, an open-loop optimal control problem is solved over a finite horizon. The computed optimal input signal is applied to the process only during the following sampling interval $[t, t+1]$. At the next time step $t+1$, a new optimal control problem based on new measurements of the state is solved over a shifted horizon.

MPC is an RHC problem where the FTCOC problem with a prediction horizon of $N$ is computed by solving online the following optimization problem:

$$
J_{t \rightarrow t+N}(x_t) = \min_{u_{t|t}, \ldots, u_{t+N-1|t}} \left[ \sum_{k=t}^{t+N-1} q(x_k|t, u_k|t) + p(x_{t+N}|t) \right]
$$

s.t.

$$
x_{k+1|t} = Ax_k|t + Bu_k|t \quad \forall k \in [t, \ldots, t+N-1] \tag{2}
$$

$$
x_k|t \in \mathcal{X}, \quad u_k|t \in \mathcal{U} \quad \forall k \in [t, \ldots, t+N-1] \tag{3}
$$

$$
x_{t+N|t} \in \mathcal{X}_f \tag{4}
$$

$$
x_{t|t} = x(t) \tag{5}
$$

where $x$ is the state and $u$ the control input. The subscript $k|t$ represents a given quantity in the prediction horizon with respect to time $t$. Equation (5) imposes the current system state to be the initial condition of the generic FTCOC problem. Equation (2) represents the discrete-time linear time-invariant system dynamics. State and input constraints are given by (3). The terminal constraint is given by (4), which forces the terminal state $x_{t+N|t}$ into some set $\mathcal{X}_f$. The stage $q(\cdot, \cdot)$ and terminal cost $p(x_{t+N|t})$ are any arbitrary, continuous, strictly positive functions.
2.2. TC-LMPC—Terminal Component Learning

This work is based on the TC-LMPC architecture first proposed in [22]. This is a reference-free iterative control strategy able to learn from previous iterations. At each iteration, the initial condition, the constraints, and the objective function do not change. The authors show how to design a terminal safe set—SS—and a terminal cost function—the Q-function—such that the following theoretical guarantees hold:

- Nonincreasing cost at each iteration;
- Recursive feasibility, i.e., state and input constraints are satisfied at iteration $j$ if they were satisfied before;
- Closed-loop equilibrium is asymptotically stable.

Considering (1), the terminal cost is given by the Q-function: $p(x_{t+N|t}) = Q^{j-1}(x_{t+N|t})$. Meanwhile, the terminal constraint corresponds to the terminal safe set $SS$: $X_f = SS^{-1}$. At the $j$th iteration, the inputs applied to the system and the corresponding state evolution are collected in the vectors given by (6) and (7), respectively.

$$u^j = [u^j_1, u^j_2, \cdots, u^j_l, \cdots]$$  (6)
$$x^j = [x^j_1, x^j_2, \cdots, x^j_l, \cdots]$$  (7)

The safe set $SS^j$, given by (8), is the collection of all state trajectories at iteration $i$ for $i \in M^j$—the set of indexes $k$ corresponding to the iterations that successfully steer the system to the final point $x_f$.

$$SS^j = \left\{ \bigcup_{i \in M^j} x^j_i \right\}$$  (8)

The $Q^j$ function, defined in (9), assigns to every point in the safe set the minimum cost-to-go along the trajectories therein.

$$\forall x \in SS^j, Q^j(x) = f^j_{x \to x_0}(x) = \sum_{k=1}^{\infty} q(x^*_{k}, u^*_{k})$$  (9)

where $i^*$ corresponds to the iteration that minimizes such a cost starting at the particular state $x$, and $t^*$ is the respective time of this state in this iteration.

The safe set works as a safe region given the shorter horizon $N$. For example, in autonomous racing, it can account for the shape of the track beyond the horizon. In this way, the controller is informed by past experimental data regarding how fast he can travel at a particular part of the track without having to compute the global optimal racing line.

2.3. Gaussian Processes Regression

Gaussian processes [29] are a non-parametric, probabilistic machine learning approach to learning in kernel machines. They provide a fully probabilistic predictive distribution, including estimates of the uncertainty of the predictions. Consider an unknown latent function $g: \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ that is identified from a collection of inputs $z_k \in \mathbb{R}^{n_x}$ and corresponding outputs $y_k \in \mathbb{R}^{n_y}$.

$$y_k = g(z_k) + w_k$$  (10)

where $w_k \sim \mathcal{N}(0, \Sigma^w)$ is independent and identically distributed Gaussian noise with diagonal variance $\Sigma^w = \text{diag}([\sigma_1^2, \cdots, \sigma_m^2])$. The set of $n$ input and output data pairs form a dictionary $D$:

$$D = \left\{ \mathbf{Y} = [y^T_1; \cdots; y^T_n], \mathbf{Z} = [z^T_1; \cdots; z^T_n] \in \mathbb{R}^{n \times n_y}, \mathbf{Z} = [z^T_1; \cdots; z^T_n] \in \mathbb{R}^{n \times n_z} \right\}$$  (11)

Assuming a Gaussian prior on $g$ in each output dimension $d \in \{1, \cdots, n_y\}$, such that they can be treated independently, the posterior distribution in dimension $d$ at an
evaluation point \( z \) has a mean and variance given by Equations (12) and (13), respectively. Further, in this situation, one refers to \( Y \) as \( y \). In other words, there is a collection of \( n \times n \)-dimensional vectors \( y^d \).

\[
\mu^d(z) = k^d_{zz}(K^d_{zz} + 1\sigma^2)^{-1} y^d
\]

\[
\Sigma^d(z) = k^d_{zz} - k^d_{zz}(K^d_{zz} + 1\sigma^2)^{-1} k^d_{zz}
\]

where \( K^d_{zz} \) is the Gramian matrix, i.e., \( [K^d_{zz}]_{ij} = k^d(z_i, z_j), [K^d_{zz}]_j = k^d(z_j, z) \), \( k^d_{zz} = (k^d_{zz})^T \), and \( k^d_{zz} = k^d(z, z) \) corresponds to the kernel function used. The specification of the prior is important because it fixes the properties of the covariance functions considered for inference—in particular, the type of kernel function \( k^d(z, z) \) used and its hyperparameters.

The multivariate Gaussian process approximation is given by \( g(z) \sim \mathcal{N}(\mu^d(z), \Sigma^d(z)) \), where \( \mu^d(z) = [\mu^1(z), \cdots; \mu^n(z)] \) and \( \Sigma^d(z) = [\Sigma^1(z); \cdots; \Sigma^n(z)] \).

### 2.4. Sparse Approximations for Gaussian Process Regression

The computational complexity of GPR strongly depends on the number of data points \( n \). In particular, a computational cost of \( \mathcal{O}(n^3) \) is incurred whenever a new training point is added to the dictionary \( D \). This is due to the need to invert \( (K^d_{zz} + 1\sigma^2) \) in Equations (12) and (13), which is a \( n \times n \) matrix. Moreover, the evaluation of the mean and variance has a complexity cost of \( \mathcal{O}(n) \) and \( \mathcal{O}(n^2) \), respectively.

Several sparse approximation techniques have been proposed to allow the application of GPR to large problems in machine learning [30]. An additional set of \( m < n \) latent variables \( \vec{g} = [g_1, \cdots, g_m] \), which are called inducing variables or support points, are used to approximate (12) and (13). These are values of the Gaussian process evaluated at the inducing inputs \( Z_{ind} = [\vec{g}_1; \cdots; \vec{g}_m] \). The latent variables are represented as \( \vec{g} \) rather than \( y \) as they are not real observations, thus not including the noise variance.

The simplest sparse approximation method is the Subset of Data (SoD) approximation, i.e., it solves (12) and (13) by substituting \( Z \) by \( Z_{ind} \). It is often used as a baseline for sparse approximations. The computational complexity is reduced to \( \mathcal{O}(nm^3) \) for training and \( \mathcal{O}(m^3) \) and \( \mathcal{O}(m^3) \) for the mean and variance, respectively.

The Fully Independent Training Conditional approximation [31] assumes that all the training data points are independent. The computational complexity is \( \mathcal{O}(nm^{3} + mn + nm^{2}) \) initially and \( \mathcal{O}(m^3) \) per test case for the predictive mean and variance, respectively. FITC can be viewed as a standard GP with a particular non-stationary covariance function parameterized by the pseudo-inputs. The mean and variance are given by

\[
\mu^d(z) = k^d_{zz_{ind}} \Theta K_{zz_{ind}z} \Lambda^{-1} y^d = k^d_{zz_{ind}} i^d
\]

\[
\Sigma^d(z) = k^d_{zz_{ind}} K_{zz_{ind}z}^{-1} K_{zz_{ind}z} - K_{zz_{ind}} \Theta K_{zz_{ind}z}
\]

where \( \Theta = (K_{zz_{ind}z} + K_{zz_{ind}} \Lambda^{-1} K_{zz_{ind}})^{-1} \), \( i^d \) is the information vector taken for each \( d \) sparse model, and \( \Lambda = \text{diag}(K_{zz} - K_{zz_{ind}} K_{zz_{ind}}^{-1} K_{zz_{ind}}) \).

### 3. Performance-Driven Controller Learning

#### 3.1. Learning Terminal Components for Autonomous Racing

Learning terminal components is considered a sub-field of controller design learning for LMPC [15]. Rosolia and Borrelli first introduced in [23] the adaptation of the core TC-LMPC architecture to the autonomous racing problem. This is formulated as a minimum time problem, where an iteration \( j \) corresponds to a lap. Therefore, the stage cost is given as follows:

\[
q(x_k, u_k) = \begin{cases} 
1 & \text{if } x_k \notin \mathcal{L} \\
0 & \text{if } x_k \in \mathcal{L}
\end{cases}
\]
where $\mathcal{L}$ is the set of points beyond the finish line. A slower trajectory contains more points until the finish line, thus having a greater cost associated.

The vehicle dynamics considers the states and input vector quantities in (17) and (18), respectively.

$$x = [s, e_y, e_\psi, v_x, v_y, r]$$ (17)
$$u = [a, \delta]$$ (18)

where $x$ collects states that describe the vehicle movement and $u$ is the vector of control inputs. $s$ is the distance along the track centerline measured from the start line; $e_y$ and $e_\psi$ are the lateral distance and heading angle errors between the vehicle pose and the centerline, respectively. These quantities are given in the curvilinear abscissa reference frame $\xi$-$\eta$—see Figure 2—also known as the Frenet reference frame. A given track is defined by the curvature $k(s)$ and maximum admissible lateral error $e_{max}(s)$ along the track centerline. $v_x$ and $v_y$ are the longitudinal and lateral vehicle velocities, respectively, while $r$ is the vehicle yaw rate. The inputs are the longitudinal acceleration $a$ and the steering angle $\delta$.

$\text{Figure 2. Local and global vehicle frames.}$

Rosolia and Borrelli [32] further extended this architecture by proposing a local TC-LMPC that significantly reduced the computational burden by using a subset of the stored data. In particular, the local convex safe set $CS_j$ is built around the candidate terminal state $c_t$ using the $N_{SS}^p$ nearest neighbours from each of the previous $N_{SS}^l$ laps. Notice that $N_{SS}^l = j - l$. These points are collected in the matrix $D_j^l$, defined in (19), which is updated at each time step. The candidate terminal state $c_t$ is the estimated value for $x_{t+N_{SS}^l}$ calculated at time $t - 1$. The approximation of the cost-to-go is computed using the costs associated with the selected states in $D_j^l$.

$$D_j^l = [x_j^l, \cdots, x_j^{N_{SS}^p}, \cdots, x_j^{N_{SS}^l}]$$ (19)

3.2. TC-LMPC for Formula Student Driverless

The original TC-LMPC architecture represents the vehicle pose in the local coordinate frame, (17). However, the discretization method used, which assumes a constant track curvature $\kappa$ within a sampling period, fails to properly describe tracks with rapidly changing curvatures. This issue was not taken into consideration in [26,32].

Therefore, we propose to change the vehicle model pose states to the global coordinate frame $X$-$Y$. In this case, $s$ and $e_y$ are still calculated because these quantities provide immediate information regarding the track, but they are not used to characterize the vehicle pose dynamics. The longitudinal control input is $P \in [-1, 1]$, which represents a pedal setpoint. This corresponds to the normalized acceleration and brake pedal travel, i.e., the
way in which the actual prototype is controlled both in simulation and reality. Thus, for our application, Equations (17) and (18), respectively, become

\[
x = [x, y, \psi, v_x, v_y, r]
\]

(20)

\[
u = [P, \delta]
\]

(21)

The adopted cost function is composed of five main parts and is given as follows:

\[
J_{t\rightarrow t+N}^{\text{LMPC}} = \min_{u(t)} \left[ \sum_{k=t}^{t+N-1} \left( (x_{k+1}|t - x_k|_t)^T Q_{\text{deriv}}(x_{k+1}|t - x_k|_t) \right) + \\
\sum_{k=t}^{t+N-2} \left( (u_{k+1}|t - u_k|_t)^T R_{\text{deriv}}(u_{k+1}|t - u_k|_t) \right) + \\
\sum_{k=t+1}^{t+N} \left( Q_{\text{el}} \cdot e_{\text{el}} + Q_{\text{quad}} \cdot e_{\text{quad}} \right) + \\
\sum_{k=t+1}^{t+N} \left( Q_{\text{elt}} \cdot e_{\text{elt}} + Q_{\text{quad}} \cdot e_{\text{quad}} \right) + \\
\sum_{k=t+1}^{t+N} \left( Q_{\text{qlane}} \cdot e_{\text{qlane}} + Q_{\text{quad}} \cdot e_{\text{quad}} \right) + \\
\sum_{i=1}^{N_{SS}} \left( Q_{\text{term cost}}(\alpha_i \times Q_i(c_i)) \right) + \\
\sum_{i=1}^{N_{SS}} \left( Q_{\text{slack}}(x_{t+N}|t - \alpha_i D_i(c_i))^2 \right) \right]
\]

(22)

First, the derivative terms apply a penalty on the squared change of a given quantity between consecutive steps along the prediction horizon, both for dynamic states and inputs. This enables us to obtain smooth trajectories. Second, a quadratic cost is applied on \(v_y\) to act as a regularization cost, which forces the vehicle into its stable domain and helps convergence. Third, a quadratic cost is applied to the lag error, which measures the accuracy of the global to local coordinate transformation [21].

The fourth part concerns the soft constraints on the states. Bounds on the states are not implemented as hard constraints since one cannot exclude that the real system moves outside the constraint range due to, for instance, model mismatch, which would render the problem infeasible [28]. Thus, the bound on a given state \(x \leq x_{\text{max}} + \epsilon\), where \(\epsilon \geq 0\), and a term \(l(\epsilon)\) is added to the cost functional. It can be shown that \(l(\epsilon) = ue + ve^2\) with a sufficiently high \(u\) and \(v > 0\) ensures that no constraint violation occurs, provided that there exists a feasible input. The bounded states are \(v_x\) and \(e_y\), where \(e_y\) needs to be bounded to stay within the track width. Additionally, a velocity ellipse is implemented to ensure that the vehicle remains within its physical limits.

Finally, the last part of the cost functional in (22) contains two penalty terms on the terminal components of the MPC. A linear cost is applied to the product of \(\alpha_i\)—the coefficients of the local safe set \(D_i\) convex hull—and \(Q_i\)—the cost-to-go of each point in the safe set, (9). Moreover, a quadratic penalty is applied to the error between the terminal state and the safe set points \(D_i\), i.e., \(x_{t+N}|t - \alpha_i D_i(c_i)\), for each point \(i\) in the safe set. While this slack term ensures that the terminal state lies within the convex hull, the terminal cost favors points in the safe set that result in faster laps.

Bounds on both inputs’ maximum level and rate are applied. These bounds may be more restrictive than the actual physical limits imposed by the robotic platform.
The control actuation $u^i_t$ applied at the current sampling time corresponds to the previous sampling time solution shifted by one step, (23). This delay aims to achieve a constant control rate given the solver’s processing time—which is significant in real time and may be inconsistent across the experiment.

$$u^i_t = u^i_j = u^i_{t|t-1}$$  

3.3. Formula Student Driverless Vehicle Model

The system dynamics in (2) results from the discretization of (24), which is the sum of an **a priori** physics-based model $f_t$ and a term to model the unknown dynamics $g_t$, given as follows:

$$\dot{x}_t = h_t(x_t, u_t) = f_t(x_t, u_t) + g_t(x_t, u_t)$$  

The pose dynamics are given by the following equations:

$$\dot{x} = v_x \cos(\psi) - v_y \sin(\psi)$$  

$$\dot{y} = v_x \sin(\psi) + v_y \cos(\psi)$$  

$$\dot{\psi} = r$$

The dynamic part of the vehicle model is modeled by a dynamic bicycle model—Figure 3. This model is frequently used in automotive control algorithms [33] and is given as follows:

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{1}{m} (F_x - F_y \sin \delta + mv_y r) \\ \frac{1}{m} (F_{R,y} + F_{F,y} \cos \delta - mv_x r) \\ \frac{1}{I_z} (F_{F,y} l_F \cos \delta - F_{R,y} l_R) \end{bmatrix}$$

where $m$ is the vehicle mass and $I_z$ is the rotational inertia about the vertical axis $z$. The front and rear axles are identified by the subscripts $a \in \{F, R\}$, respectively. $l_a$ is the distance between the vehicle center of gravity and the corresponding axle. The lateral force $F_{a,y}$ is given as

$$F_{a,y} = -2D_a \sin (C_a \arctan(B_a \alpha_a))$$

where the tire coefficients $B_a$, $C_a$, and $D_a$ are experimentally identified and the slip angle $\alpha_a$—the angle between the tire centerline and its velocity vector—is computed as follows:

$$\alpha_F = \arctan \left( \frac{v_y + l_F r}{v_x} \right) - \delta$$

$$\alpha_R = \arctan \left( \frac{v_y - l_R r}{v_x} \right)$$

The longitudinal force $F_x$ is given as follows:

$$F_x = 2 \frac{T_{\text{max}} \times \phi \times P}{r_{\text{wheel}}} - C_{\text{roll}} \times m \times g + \frac{1}{2} \rho \times C_d \times A_f \times v_x^2$$

where $T_{\text{max}}$ is the maximum available torque at each of the two rear-axle in-wheel motors. $\phi$ is the transmission gear ratio and $C_{\text{roll}}$ is the roll resistance factor. Concerning the aerodynamic drag force, $\rho$ is the air density, $A_f$ is the vehicle frontal area used as a reference for the force calculation, and $C_d$ is the drag coefficient.
Figure 3. Dynamic bicycle model (reprinted from [34]). The red arrows represent forces applied in the vehicle. The blue arrows represent both linear and angular velocities. The green arrows represent the position vector of the vehicle’s center of gravity with respect to the global coordinate frame.

4. System Dynamics Learning

Gaussian process regression is used to predict the error between the vehicle model—Equations (25)–(28)—and the available measurements, i.e., estimate $g_t$ in Equation (24). We do not make use of the uncertainty prediction. Thus, Equations (13) and (15) are disregarded in our application.

We assume that the modeling error only affects the dynamic part of the first-principle model [20], i.e., the velocity states. Therefore, the training outputs—(10)—are given by the difference in the velocity components between the measurement $x_{k+1}$ and the nominal model prediction:

$$y_k = B_d^\dagger (x_{k+1} - f(x_k, u_k))$$

(33)

where $B_d^\dagger$—the Moore–Penrose pseudo-inverse of $B_d = [0_{3 \times 3}; I_{3 \times 3}]$—is a selection matrix introduced to consider only the velocity components. Hence, GPR predicts the error in the three velocities: $d \in \{e_x, e_y, e_r\}$, and $n_g = 3$.

The GPR training input, i.e., the feature state, is $z = \{v_x; r; P; \delta\}$. This is based on the assumption that the model errors are independent of the vehicle position. This disregards potential modeling shortcomings introduced by, for instance, segments of the track that have different traction conditions, e.g., due to puddles. Further, we removed $v_y$ as we identified a strong correlation between this quantity and $r$. This is not surprising as both quantities characterize the lateral movement of the vehicle. The removal of $v_y$ instead of $r$ is justified by the fact that it is difficult to precisely estimate $v_y$, while $r$ is measured directly using a gyroscope. These approximations substantially reduce the learning problem’s dimensionality.

The covariance function—$k_{zz}^d$ in (12) and (13)—used is the squared-exponential kernel with the independent measurement noise component $\sigma^2_{n_d, d} \delta z^2$:

$$k_{zz}^d(z, \bar{z}) = \sigma^2_f d \exp \left( -\frac{1}{2} \frac{(z - \bar{z})^T(z - \bar{z})}{\lambda^2_d} \right) + \sigma^2_{n_d, d} \delta z^2$$

(34)

Two sparse approximations have been explored: (i) SoD and (ii) FITC. The first approximation method is essentially a full GP that does not use all data available. This means that the computations for the error prediction $g_t$ in (24) are those of an exact GP given by (12). Meanwhile, for the second approach, the computations are those of (14).
The SoD approximation is often seen as naive as only a fraction of the available dataset is used. Hence, in order to improve the probability of good performance, rather than selecting the \( m \) points randomly, we use a technique to select points included in the active set that maximizes the information gain criterion \([27]\).

Finally, it should be noted that some quantities can be pre-computed that otherwise could prevent real-time feasibility. In particular, \((K_{ZZ}^{-1} + I\sigma_{d}^{2})^{-1}y_d\) in (12) only needs to be recomputed whenever the training data \( D \) are changed—this is the training step. In the SoD offline learning case, it is only computed once before the controller is launched. Inference corresponds to the rest of the computation of (12).

In the FITC approximation, the training step corresponds to the determination of the information vector \( \mathbf{r}^{d} \), which only requires recomputation when either the training dataset \( D \) or the inducing points \( Z_{\text{ind}} \) are changed. In our application, however, the inducing points are updated at each sampling time. They are equally distributed along the last sampling time’s shifted predicted trajectory. This is a sensible placement of the inducing points since the new test cases are expected to be near the preceding ones, as the trajectory does not change significantly at consecutive sampling times. This means that the online adaptation of the dictionary \( D \) is immediately possible.

GPR is generally susceptible to outliers, which can hinder the model error learning performance \([20]\). Moreover, large and sudden changes in the GPR predictions can lead to erratic driving behavior. To attenuate these effects, we only include data points in the dictionary \( D \) (both online and offline) whose measurements fall within predefined bounds \( \pm y_{\text{lim}} \), defined from physical considerations and empirical knowledge.

The hyperparameters are tuned offline based on pre-collected data. There are two main reasons that this is not done online. First, this optimization is not real-time feasible. Second, it is assumed that the general trend of the model error remains constant throughout the vehicle’s operation.

5. Implementation and Results

5.1. Implementation

Figure 4 exhibits the autonomous racing software architecture used by the FST Lisboa team. In the Trackdrive event, the entire control module in Figure 4 (green nodes) is substituted by the TC-LMPC algorithm.

![Figure 4. FST10d autonomous racing software stack.](image)

FSSIM (https://github.com/AMZ-Driverless/fssim (accessed on 20 April 2023)) is the vehicle simulator used. AMZ Driverless developed this vehicle simulator, dedicated to the FSD competition, and released it open-source to other teams. This team reported 1% lap time simulator accuracy compared with their actual FSG 2018 10-lap trackdrive run \([34]\). Due to real-time requirements, this simulator does not simulate raw sensor data, e.g., camera or LiDAR data. Instead, cone observations around the vehicle are simulated using a given cone sensor model. This means that, in simulation, the perception module in Figure 4 (red nodes) is not the same as in actual prototype testing.
We developed a custom ROS/C++ implementation of the TC-LMPC architecture using FORCESPRO [24,25]—a solver designed for the embedded solving of MPC—to solve the optimization problem. The computations associated with GPR model error prediction described in Section 4 resort to the C++ open-source albatross (https://swiftnav-albatross.readthedocs.io/en/latest/index.html (accessed on 20 April 2023)) library developed by Swift Navigation.

5.2. Model Learning Analysis

In this section, we investigate the performance of the sparse GPR approximations for the prediction of the model error. We use the compound average error to evaluate the overall model learning performance: $||e_{nom}||$ is the average 2-norm error of the nominal model, i.e., $||e_{nom}|| = ||B^* D(x_{k+1} - f(x_k, u_k))||$, and $||e_{GP}||$ is the corresponding error of the corrected dynamics, i.e., $||e_{GP}|| = ||B^* D(x_{k+1} - f(x_k, u_k) - g(z_k))||$. The results shown here correspond to the average over 10 laps. To evaluate the effectiveness of the proposed solution, the results shown were collected using FST Lisboa’s—the FSD team from the University of Lisbon—autonomous vehicle parameters in the MPC dynamic vehicle model, while using the AMZ Driverless car model in the simulator.

For the SoD approximation, we use the active set selection method described in Section 4 to find the active set of different sizes $m_{SoD}$ for each model $d$ from a dictionary of $n_{SoD} = 43,329$, collected over 10 different runs with changing parameters. Moreover, we use the active set of the SoD approximation as the training set for the FITC approximation, i.e., $n_{FITC} = m_{SoD}$.

Table 1 shows the model error prediction fitness for both sparse GPR approximations. For the SoD results, on the left, the horizontal dashed line separates the results of the offline (above) and online (below) schemes. For the online FITC results, on the right, the horizontal dashed line separates the results of $n_{FITC} = 300$ (above) and $n_{FITC} = 400$ (below) schemes.

| $m_{SoD}$ | $||e_{nom}||$ | $||e_{GP}||$ | $m_{FITC}$ | $||e_{nom}||$ | $||e_{GP}||$ |
|---|---|---|---|---|---|
| 300 | 0.26 | 0.08 | 10 | 0.26 | 0.17 |
| 600 | 0.25 | 0.07 | 20 | 0.26 | 0.15 |
| 200 | 0.27 | 0.09 | 10 | 0.27 | 0.12 |
| 300 | 0.27 | 0.07 | 20 | 0.27 | 0.13 |

We conclude that there is a positive correlation between $m_{SoD}$ and the model error fitting ability. In the offline fashion, the 2-norm average error reduction, i.e., reduction from $||e_{nom}||$ to $||e_{GP}||$, is 63.9% and 70.7% with $m_{SoD} = 300$ and $m_{SoD} = 600$, respectively. The computation cost evolves linearly with $m$ but, within this range, takes acceptable values given the current node rate of 20 Hz. Arguably, we could further increase $m_{SoD}$ but likely with negligible model learning improvements.

Instead, we should aim to adapt the training data online as this would enable adaptation to changing conditions or even simply the collection of data from dynamic maneuvers not included in the original dictionary. With a dictionary size of $m_{SoD} \leq 300$ data points, the model error reduction is 65.4% and 75.7% for $m_{SoD} = 200$ and $m_{SoD} = 300$, respectively. In particular, it enables more aggressive maneuvers toward the end of the event while keeping the corrected dynamics error relatively low.

In the online case, the model with $m_{SoD} = 300$ performs significantly better than the model with $m_{SoD} = 200$. While the average node processing time is well within the limits, it too often breaks the real-time requirement.

As explained in Section 4, the FITC sparse approximation is a natural candidate to enable online learning. The best-performing model with $m_{FITC} = 10$ yields model
learning performance comparable to that of the SoD approximation. However, the corrected
dynamics average error is 0.12, slightly above the SoD benchmark value of 0.09.

5.3. Simulation Results—Model Mismatch Influence

In Table 2, we show the lap times along the 10-lap track drive event and the average
model error for the FSG track. The initial safe set was collected using a pure pursuit
controller [35] following the track centerline. It is composed of four laps with lap times
of around 28.80 s. If an initial safe set containing faster trajectories is collected, the initial
laps would be faster but would eventually converge to approximately the same lap time.
The controllers herein have a prediction horizon of \( N = 20 \), which corresponds to a
look-ahead time of 1 s, unless stated otherwise. The TC-LMPC results prove the iterative
improvement nature of the architecture. The first lap is immediately 33% faster compared to
the path-following controller. Equivalently, the last lap is 39% faster. Furthermore, the last
lap corresponds to a 10% improvement compared to the first TC-LMPC lap.

Table 2. FSG lap times and model error.

<table>
<thead>
<tr>
<th>Lap</th>
<th>TC-LMPC Time [s]</th>
<th>( | e_{nom} | )</th>
<th>TC-LMPC Time [s]</th>
<th>( | e_{nom} | )</th>
<th>( | e_{GP} | )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19.36</td>
<td>0.22</td>
<td>18.92</td>
<td>0.21</td>
<td>0.06</td>
</tr>
<tr>
<td>2</td>
<td>19.33</td>
<td>0.21</td>
<td>18.75</td>
<td>0.21</td>
<td>0.07</td>
</tr>
<tr>
<td>3</td>
<td>19.34</td>
<td>0.21</td>
<td>18.73</td>
<td>0.21</td>
<td>0.07</td>
</tr>
<tr>
<td>4</td>
<td>19.33</td>
<td>0.22</td>
<td>18.71</td>
<td>0.21</td>
<td>0.07</td>
</tr>
<tr>
<td>5</td>
<td>17.78</td>
<td>0.29</td>
<td>17.17</td>
<td>0.29</td>
<td>0.08</td>
</tr>
<tr>
<td>6</td>
<td>17.45</td>
<td>0.30</td>
<td>16.85</td>
<td>0.31</td>
<td>0.11</td>
</tr>
<tr>
<td>7</td>
<td>17.44</td>
<td>0.31</td>
<td>16.77</td>
<td>0.32</td>
<td>0.13</td>
</tr>
<tr>
<td>8</td>
<td>17.51</td>
<td>0.29</td>
<td>16.84</td>
<td>0.32</td>
<td>0.11</td>
</tr>
<tr>
<td>9</td>
<td>17.43</td>
<td>0.30</td>
<td>16.80</td>
<td>0.31</td>
<td>0.12</td>
</tr>
<tr>
<td>10</td>
<td>17.43</td>
<td>0.30</td>
<td>16.96</td>
<td>0.32</td>
<td>0.13</td>
</tr>
</tbody>
</table>

Figure 5 shows the 10-lap FSG track drive trajectories using the TC-LMPC strategy.
The finish line is at the origin and the vehicle runs clockwise. It can be seen that TC-LMPC
exploits the track layout to improve the vehicle’s performance. Nevertheless, the third
column of Table 2 shows a severe model mismatch, which causes the vehicle to violate the
track constraint. See, for instance, the exit of the hairpin, where the trajectory is on top of
the track boundary. This indicates that a cone was hit, since the trajectory corresponds to
the center of mass. Furthermore, the approach to the slalom segment is not considered
optimal by empirical vehicle dynamics standards. The vehicle brakes too late, which leads
to a slower slalom with greater steering actuation required.

We now analyze the performance of the controller when the GPR model learning
scheme is deployed. Table 2—TC-LMPCML—shows that the last lap is 41% and 10% faster
when compared to the pre-collected path-following lap and the first lap, respectively.
Furthermore, the total event time, 176.5 s, is 5.9 s faster than when using TC-LMPC, a 3%
improvement. These results correspond to the online SoD model with \( m_{SoD} = 200 \). Figure 6
shows the corresponding 10-lap FSG track drive trajectories. It is clear that the reduced
model mismatch prevented the vehicle from violating the track constraint. However, it
seems that there is still room for improvement during the slalom segment.
Figure 5. TC−LMPC FSG trackdrive trajectory \( N = 20 \). The grey arrow shows the track direction.

Figure 6. TC−LMPC\textsubscript{ML} FSG trackdrive trajectory \( N = 20 \). The grey arrow shows the track direction.

5.4. Simulation Results—Prediction Horizon Influence

We subsequently tested TC-LMPC\textsubscript{ML} with increasing prediction horizons. In Table 3, we display the lap times and modeling errors for two sets of controller gains with \( N = 30 \) or a look-ahead time of 1.5 s. The results on the left correspond to the parameters used thus far in this section. For the controller on the right, we reduced some derivative costs and the regularization cost on \( v_y \) and increased the \( Q \)-function-associated cost to promote greater track progress at each sampling time.

With a longer horizon, both controllers achieve optimal trajectories in the slalom segment and can safely navigate the vehicle around the track such that the safe set loses
its relative importance. Specifically, the controllers are able to predict consistently until the slowest point on a given corner. In this way, the information conveyed by the safe set regarding the types of maneuvers that follow is not as valuable. The safety characteristic only applies when model learning is deployed. Otherwise, the severe model mismatch hinders the performance. This is substantiated by the fact that both controllers achieve small lap times in the first few laps and quickly converge to their steady-state lap times of around 16.75 s for the default controller and 16.16 s for the aggressive controller.

Table 3. FSG lap times and model error \([N = 30]\).

| Lap | Default Parameters | | Aggressive Parameters | | |
|-----|---------------------|-----------------|------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|
|     | Time [s]            | \(\varepsilon_{\text{nom}}\) | \(\varepsilon_{\text{GP}}\) | Time [s]            | \(\varepsilon_{\text{nom}}\) | \(\varepsilon_{\text{GP}}\) |
| 1   | 17.16               | 0.27            | 0.07                   | 16.37             | 0.39            | 0.14            |
| 2   | 16.92               | 0.26            | 0.07                   | 16.18             | 0.37            | 0.15            |
| 3   | 16.91               | 0.26            | 0.07                   | 16.17             | 0.37            | 0.15            |
| 4   | 16.88               | 0.26            | 0.07                   | 16.13             | 0.37            | 0.15            |
| 5   | 16.69               | 0.28            | 0.08                   | 16.17             | 0.36            | 0.15            |
| 6   | 16.66               | 0.28            | 0.09                   | 16.21             | 0.37            | 0.15            |
| 7   | 16.67               | 0.28            | 0.08                   | 16.19             | 0.36            | 0.15            |
| 8   | 16.68               | 0.28            | 0.08                   | 16.14             | 0.37            | 0.16            |
| 9   | 16.68               | 0.27            | 0.08                   | 16.15             | 0.37            | 0.16            |
| 10  | 16.69               | 0.27            | 0.08                   | 16.11             | 0.36            | 0.15            |

Both models used the offline version of the SoD approximation with \(m_{\text{SoD}} = 600\). The model learning scheme is able to significantly reduce the model mismatch to enable safe aggressive racing. For the first case, with default parameters, the corrected dynamics model average error is around 0.08, which corresponds to a reduction of about 70% compared to the nominal model. Meanwhile, for the aggressive controller, the final model mismatch is on average 0.15, a reduction of 60%. The model learning scheme can achieve an acceptable value of corrected dynamics model mismatch, even in the case of offline model learning on the aggressive controller. It should be noted that such high nominal model errors were not present in the original training dataset.

6. Conclusions and Future Work

This paper presented the extension of the results of the TC-LMPC architecture to the autonomous electric racing FSD context. We have demonstrated that the TC-LMPC using a dynamic bicycle model with an appropriate terminal safe set and \(Q\)-function leads to safe iterative improvements. Despite the significant model mismatch, the best lap time is reduced by 10%, when compared to the first lap using the same controller, at the 10-lap FS event. When using GPR with the active training set selection method introduced for model learning, we are able to successfully predict the modeling error, which leads to a 5.9 s total time reduction for the 10-lap FSD event.

The main limitation of the proposed solution, inherent in the use of model-based controllers, is model mismatch. This can impact the feasibility of the solution, and the theoretical guarantees of the TC-LMPC listed above might not hold. The other major limitation is that this architecture requires pre-collected data, which requires a suboptimal but safe controller. Alternatively, replication of this work may be possible by resorting to manual data collection, which should be available in most practical applications.

These simulation results are promising for an experimental implementation on the full-sized FST10d electric racing prototype. An important focus of the research on learning model-based controllers has been on reducing model mismatch. An implementation of an online dictionary management procedure \([20]\) is under study. Further, research work is underway to test the Automatic Relevance Determination GPR kernel and other ML techniques, such as the use of Bayesian linear regression or neural networks. The minimum-time stage cost for the TC-LMPC \(Q\)-function could be augmented such that points that yield
long-term benefits are more favored. We are also studying the use of the state uncertainty measurement, which is inherent with the application of Gaussian processes, in a robust learning-based model predictive control framework with constraint tightening. Finally, the automatic adjustment of the MPC parameters using a reward function exploited by a reinforcement learning algorithm is also a future line of research.

Author Contributions: Conceptualization, J.P., G.C., P.U.L. and M.A.B.; methodology, J.P., G.C., P.U.L. and M.A.B.; software, J.P. and G.C.; validation, P.U.L. and M.A.B.; writing—original draft preparation, J.P.; writing—review and editing, P.U.L. and M.A.B.; visualization, J.P.; supervision, P.U.L. and M.A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the Portuguese Science and Technology Foundation (FCT), through IDMEC, under LAETA, project UID/EMS/50022/2020, and LARSyS strategic funding—FCT Project UID/50009/2020.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations
The following abbreviations are used in this manuscript:

AD         Autonomous Driving  
MPC        Model Predictive Control  
FSD        Formula Student Driverless  
ML         Machine Learning  
LMPC       Learning-Based Model Predictive Control  
GPR        Gaussian Process Regression  
BLR        Bayesian Linear Regression  
MPCC       Model Predictive Contouring Control  
TC-LMPC    Terminal Component Learning-Based Model Predictive Control  
FITC       Fully Independent Training Conditional  
RHC        Receding Horizon Control  
FTCOC      Finite-Time Constrained Optimal Control  
SoD        Subset of Data

References


Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.